Disparity between the Programmatic Views and the User Perceptions of Mobile Apps

Nahida Sultana Chowdhury
Department of Computer and Information Science
Indiana University-Purdue University Indianapolis
Indianapolis, IN, USA
nschowdh@iupui.edu

Rajeev R. Raje
Department of Computer and Information Science
Indiana University-Purdue University Indianapolis
Indianapolis, IN, USA
rraje@cs.iupui.edu

***Abstract*— User perception in any mobile-app ecosystem, is represented as user ratings of apps. Unfortunately, the user ratings are often biased and do not reflect the actual usability of an app. To address the challenges associated with selection and ranking of apps, we need to use a comprehensive and holistic view about the behavior of an app. In this paper, we present and evaluate Trust based Rating and Ranking (TRR) approach. It relies solely on an apps' internal view that uses programmatic artifacts. We compute a trust tuple (Belief, Disbelief, Uncertainty – B, D, U) for each app based on the internal view and use it to rank the order apps offering similar functionality. Apps used for empirically evaluating the TRR approach are collected from the Google Play Store. Our experiments compare the TRR ranking with the user review-based ranking present in the Google Play Store. Although, there are disparities between the two rankings, a slightly deeper investigation indicates an underlying similarity between the two alternatives.**

Keywords*— *Subjective Logic; Online Marketplace; Apps; User Rating; Trust; Evidences

I. INTRODUCTION

The numbers of smart computing devices and their users have increased at a fast pace in the global market. This proliferation has led to an incredible growth in the number of apps for these devices. For example, in March 2017, the number of apps available for download in leading two app stores (Android and Apple's app store) was 5 million [1]. This number is expected to increase in the future. As there are many apps offering similar services, the users carry out manual attempts made to select the "best" app for their specific needs. Such a manual exploration makes the selection process laborious and challenging. Some apps (e.g., WhatsApp), which do not have good peer alternatives, are easy to choose. However, for many other categories (e.g., photography), there are very few systematic approaches that assess and help the users to choose the best app for their needs. These approaches include the monitoring the top lists, read reviews, and experiment with features. In most cases, the features associated with an app are the number of downloads, installs/updates, number of ratings, average rating score, content and sentiment analysis of reviews. At present, Android and Apple's app stores both use the rating numbers to make decisions about promoting an app.

A typical user considers the textual reviews and rating scores as the only measures while selecting an app; unfortunately, the user ratings are often biased do not reflect the actual usability of an app. As indicated in [2], the sole reliance on reviews and ratings is not suitable due to the various reasons such as: ratings suffer from self-selection bias; poorly written reviews; self-promotion of apps by the companies and developers' requesting friends to give poor ratings to competing apps. To address these challenges associated with selection and ranking of apps, we need to use a holistic view about the behavior of an app. past, we have defined the trust of an app (or a software service)[1] as its conformance to its specification [3] and proposed two views of a service: an internal view or the programmatic view[2] that uses programmatic artifacts (e.g., system source code, specifications, etc.), and an external view that uses the non-programmatic artifacts (e.g., user ratings and reviews in public marketplaces). In our past work ([4, 5]), we have focused on the external view to quantify the trust of an app using evidence-based techniques (such as theory of belief [6], and associated NLP schemes [7]). The trust of an app is represented as a tuple of belief, disbelief and uncertainty (B, D, U). In this paper, we explore the internal views of publically available apps by applying the principles of static code analysis (via the FindBugs tool) and generate internal-evidences of apps. These evidences then are used to create an internal trust tuple (B, D, U). We use these internal tuples to rank order apps offering similar functionality and compare our ranking with the ranking offered by the app stores.

This rest of the paper is organized as follows: Section 2 covers the related literature on different approaches used to quantify the trust of mobile apps before or after their development. Section 3 explains the proposed approach in detail. Section 4 discusses the experimental results. Section 5 summarizes the insight gained and concludes with future work plan.

II. RELATED WORKS

There are many approaches for predicting bugs inside a mobile application via testing techniques – e.g., functional testing. Espresso [8], provided by Google, is capable of solving the concurrency issues. However, it runs on an emulator resulting in limited performance issues (such as

[1] In this paper, we have used the words "app" and "service" interchangeably.
[2] In this paper, we have used the words "internal view" and "programmatic view" interchangeably.

This is the author's manuscript of the article published in final edited form as:
Chowdhury, N. S., & Raje, R. R. (2017). Disparity between the programmatic views and the user perceptions of mobile apps. In 2017 20th International Conference of Computer and Information Technology (ICCIT) (pp. 1–6). https://doi.org/10.1109/ICCITECHN.2017.8281774

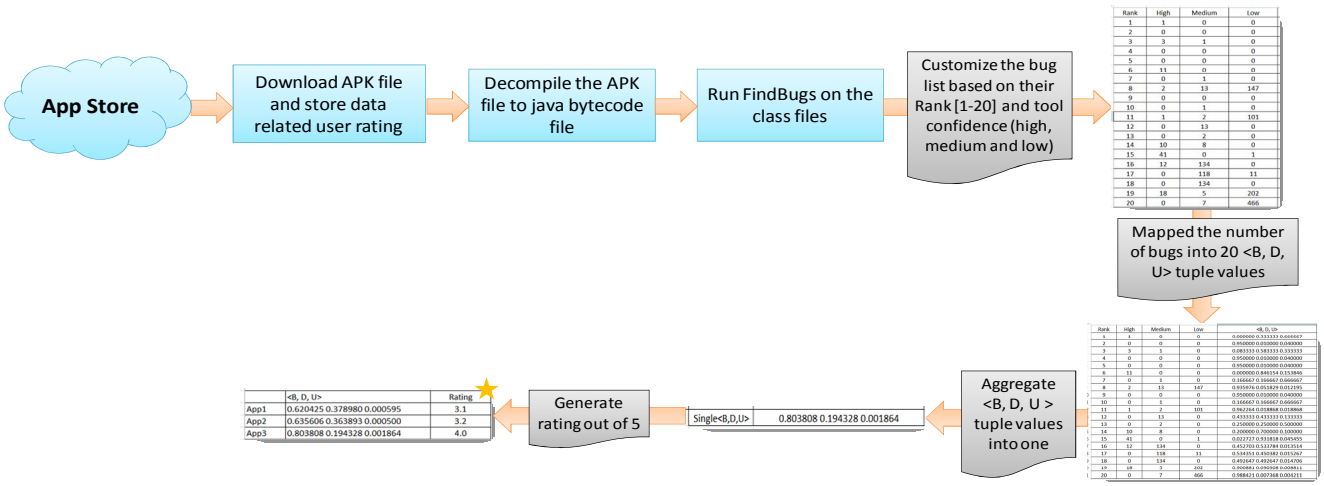

Fig. 1. Our Approach

memory constraint, display screen size, etc.). Another useful tool is Monkey [9], which comes with the Android developers' toolkit. It can only generate UI events where users have to specify the desired number of events. An automated testing tool (Bug Rocket) provided by Ma et al. [10] associates distributed testing environments with testing automation based on reverse engineering techniques. Other tools to test mobile applications include Dynodroid [11], EvoDroid [12] and SwiftHand [13].

Static code analysis tools are capable of detecting possible runtime errors (e.g., dereferencing null pointer), logical inconsistency, and security violations (e.g., SQL injection) in an app. This analysis can take place at different levels such as the source code level, the binary code level, and the bytecode level. FindBugs [14] and Jlint [15] are open-source static bytecode analyzers for Java. FindBugs is capable of covering more bug types than Jlint (e.g., unreachable code). Hammad et al. [16] used FindBugs to determine which categories of bugs occurred more frequently in low rated apps rather than in high rated apps by examining the relationships between each category of bugs in an app and the corresponding app rating. In our work, we have used FindBugs to identify different categories of bugs in terms of bug ranks (1-20) and bug confidence levels (such as high, medium and low). The bug rank represents the severity of the bug and the confidence level indicates confidence of the tool regarding the bug existences.

Several efforts have quantified the trust of a service based external views. In [17], authors present a study that investigates to what extent NLP, Sentiment Analysis [18] and Text Analysis features support to identify app store reviews relevant for the maintenance and evolution of mobile apps. Palomba et al. [19] reveal how developers address user reviews to improve their apps' success in terms of ratings. Gallege et al. ([4, 5]) have quantified trust values of services using publically available external evidences in forms of user reviews. In our proposed approach, we have used a similar quantification using internal evidences generated by FindBugs.

III. Proposed Approach

When an Android user needs an app, usually she searches the Google play store; where there are plenty of choices offering similar functionality. In most cases, she would simply choose the highest rated app (using the in-built star rating) from the suggested list. This five star rating system is questionable, as in the most of the cases a user provides either a five star or a one star rating [20] based on a positive experience or any problems encountered during installation or usage. Such a process may not reflect the quality and the trust of an app accurately. Therefore, to select a good quality app will require more details than the user ratings. Below we discuss our approach that collects internal evidences about an app by using FindBugs. First we briefly present background about FindBugs, principles of Subjective Logic, and prevalent rank ordering practices – all of which are used in our approach.

A. Static Analysis Tool: FindBugs

While there are many open-source static analysis tools, we chose FindBugs as it is capable of reducing the number of false positive warnings [21]. Moreover, FindBugs can perform its analysis on bytecode rather than source code. As we do not have an access to the unpackaged source code of published apps, we can easily run this tool on Jar files to detect occurrences of bug patterns. FindBugs is capable to identify over 400 possible bug patterns. These bug patterns are categorized into the following list: Bad Practice, Malicious Code Vulnerability, Multithreaded Correctness, Dodgy Code, Correctness, Performance, Internationalization, Experimental and Security [14]. FindBugs also assigns different priorities (from 1 to 20) to each bug related warning; where 1 represents the top priority bug and 20 represents the lowest priority bug. The priority level of the warnings is dependent on how confident (high, medium, and low) the tool is regarding the presence of that bug. High confidence means that the identified bug is certainly a real bug. Low confidence bugs are ideally false positives and medium confidence bugs lie in between these two extremes. As FindBugs has a relatively low

percentage of false positives, most of the bugs it finds are valid bugs [22].

B. Subjective Logic

Subjective logic is a form of probabilistic logic, created by Jøsang [23], has been used extensively in trust management, and system evaluation. In this model, trust is seen as a balance between belief, disbelief and uncertainty and is represented as a tuple of <B, D, U>; (belief, disbelief, uncertainty). For a single opinion about a proposition, the sum of b, d, and u is 1. There are three special tuples in the model: full belief, B = (1,0,0); full disbelief, D = (0,1,0); and full uncertainty, U = (0,0,1). Each evidence can be used to compute the <B, D, U> tuples. For our study, each internal evidence is generated by FindBugs is used to compute the trust tuples associated with that app. Presence of well-defined operators (e.g., conjunction, disjunction, negation, recommendation, ordering, and consensus [24]) is the main strength of Jøsang's model. These operators are used to evaluate, aggregate, and compare trust values. In our approach, we have used the consensus and ordering operators.

C. Popular practices for App Ratings

Five most popular app-stores (Apple's App Store, Google Play Store, Amazon Appstore, Windows Phone Store, and Blackberry AppWorld) use simple rating mechanisms known as the store rating. The store rating of an app is represented as a number of stars from 1 to 5, and is aggregated from individual user ratings. For example, in the Google Play Store, the store rating of an app is the cumulative average of all individual user ratings over all the versions.

D. Proposed Approach

Our approach, shown in Figure 1, encompasses different phases. First, we picked the Google Play Store as the target app store. In addition to its popularity, we chose the Google Play Store as most of the android apps are developed in Java and their bytecode can be analyzed by FindBugs. Then we identified the top 10 categories in Google Play – which are education, lifestyle, entertainment, business, personalization, tools, music & audio, books & reference, travel & local, and puzzles [25]. From each category, we selected three apps and stored their related data such as APK (Android Package Kit) file and the corresponding user rating. Then we decompiled the APK files to retrieve the Java bytecode files (class files). We, then, passed the class files to FindBugs as input to analyze the existence of bugs inside an app. Finally, we used these evidence to compute the <B, D, U> tuples and order apps.

1) Evaluation of <B, D, U> tuples: As indicated earlier, we define trust of an app as its ability to deliver its functionality as indicated by its specification. Any evidence that suggests such as conformance is a positive evidence and one that suggests a violation is a negative evidence. In our approach, we consider the presence of high confidence bugs (indicated by FindBugs) as negative evidences and the low confidence bugs as positive evidences. The medium confidence bugs are considered as uncertain evidences, which are equally distributed between the positive and negative evidences. More precisely, the computations of B, D, U values, to quantify the internal view of the trust of an app, are carried as follows [26]:

$$b = (\text{number of positive evidences}) / (\text{total evidences} + n) \quad (1)$$

$$d = (\text{number of negative evidences}) / (\text{total evidences} + n) \quad (2)$$

$$u = n / (\text{total evidences} + n) \quad (3)$$

Here 'n' indicates the number of possible outcomes. In our case, n is 2, as a bug is either present or absent. For example, if for a particular bug rank, we received 41 high confidence bugs, 6 medium confidence bugs and 256 low confidence bugs then using the above formulae, the trust tuple will be <0.144, 0.85, 0.006>.

2) Aggregate tuple values into single <B, D, U>: As indicated earlier, FindBugs produces twenty possible categories of evidences for an app where each category indicates a different bug priority. Using the above mentioned formula, for each app, we compute twenty <B, D, U> tuples. As these tuples indicate different opinions about the trust of the same app, these can be combined by the use of the consensus operator to create a single <B, D, U> tuple. The default consensus operator, suggested by Jøsang, treats fused opinions equally, which makes it difficult to deal with the weighted opinions. Zhou et al. [27] have proposed two fusion operators (cumulative weighted fusion operator and averaging weighted fusion operator) that are capable of dealing with fusing opinions according to their weights in a fair way. As in our approach, all the opinions are independent, we have applied the cumulative weighted fusion operator to combine individual tuples into a single tuple. Once we have such single tuples for similar apps, these apps can be ordered using the ordering operator, which uses the notion of probability expectancy. Our algorithm is described below. Please note that we normalize our rating to be out of 5 to match the rating used by the Google Play Store – such a normalization allows the comparison of our rankings with the Play Store's rankings.

Trust rating and ranking (TRR) algorithm

Input: *Evidences generated by FindBugs*
For each App{
  For each Evidence (1 to 20){
    Calculate <B, D, U> tuples using formulae 1 to 3;
  }
  Aggregate trust tuples by the use of the weighted consensus operator and generate a Single <B, D, U> tuple;
}
Use ordering operator to generate the app rating and normalize the rating to be out of 5;

Based on the rating order them 1 to N;

Output: *Ordered Ranking based on internal evidence-based rating Apps*

IV. Results and Discussion

We empirically evaluated this approach by applying it to the Android Marketplace. We picked 60 free apps from top 10 categories of the Google Play Store (six apps from each

category). Based on their functionality, we split these 60 apps into two groups of 30 app each. The first group contained "homogeneous apps" (i.e., apps offering very similar functionality) and the second group consisted of "heterogeneous apps" (i.e., apps belonging to the same top-level category, such as puzzles, but offering different puzzle games). We applied FindBugs to both these groups; collected internal evidences and fed them to our algorithm. We also used a Java decompiler to reproduce Java source code from bytecode. Using a static code analysis tool, we extracted the total number of source code lines (excluding comment lines). The total number of bugs for each app, identified by FindBugs, and the number of lines extracted were used to compute BUGS/KLOC (KLOC = thousands of lines of code) for each app. This ratio, along with the Store ranking based on user reviews, provide us with two traditional ways of ranking apps. The rankings generated by our algorithm were compared, using the Kendall Tau Distance method [28], with store's default rakings based on user reviews and the traditional rakings based on the number of bugs per KLOC. Distances of 0% and 100% indicate identical and opposite rankings respectively. Below we discuss the results of our experiments.

A. Heterogeneous Apps

Table I indicates the differences between the orderings based on user ratings and traditional approach for heterogeneous apps. Only in one case, the orderings are same while in two cases, the orderings are opposite.

Table II indicates the differences between the orderings based on the user ratings and our approach for the same heterogeneous apps. For three categories, the ordering is opposite; while for two categories, it is same. In four of the remaining five categories, the ranking is closer (33% mismatch) to the user-based ratings, while for the last category, it is farther (66%) from the user-based ratings.

B. Homogeneous Apps

Table III indicates the results in case of homogeneous apps. Unlike the heterogeneous case (Table II), the dissimilarity is more prominent in the user review-based ratings and the TRR approach. Eight rankings are dissimilar between the two schemes, while only two rankings are similar.

C. Discussion

For 30 homogeneous apps from top ten categories, we found six apps that have opposite orderings (based on the user ratings) when compared with the ordering using our TRR method. These six cases are grouped into two following categories:

Table I: DISTANCE BETWEEN ORDERING BASED ON USER RATING AND TRADITIONAL APPROACH

App category	Distance
Education	100%
Entertainment	33%
Business	33%
Books & References	33%
LifeStyle	33%
Music & audio	66%
Personalization	100%
Puzzles	66%
Tools	66%
Travel & local	0%

Table II: DISTANCE BETWEEN ORDERING BASED ON USER RATING AND TRR APPROACH – HETEROGENEOUS APPS

App category	Distance
Education	100%
Entertainment	33%
Business	66%
Books & References	33%
LifeStyle	33%
Music & audio	33%
Personalization	100%
Puzzles	0%
Tools	0%
Travel & local	100%

Good to Bad: In this case, the users rated the app as the topmost and the TRR algorithm ranked it last. A few sample supportive (first two comments) and critical user comments (last two comments) for such a case from the category of Education include:

- "Its totally free and awesome app...it provide a lot of information and is add free.. The challenges are one of best thing."
- "This app is very easy, interactive, user friendly and resourceful. Thank you developer."
- "**Shortcut quizzes are not working**, quite frustrating as I already know another programming language, and yet I need to waste time on basics. Another poor thing is that questions asked are extremely basic and poorly test actual knowledge learnt. This app is only good if you already know what you are approximately doing."
- "Nice app... I absolutely liked it **until Data types, arrays and pointer option started evoking unknown errors**... Whenever I click on it, app suddenly closed.. Kindly resolve it... Thank you."

As seen from above, the supportive reviews seem to focus on non-functional aspects (e.g., being free or user friendly-ness), while the critical reviews seem to highlight the issues

Table III: DISTANCE BETWEEN ORDERING BASED ON USER RATING AND TRR APPROACH – HOMOGENEOUS APPS

App category	Distance
Education	66%
Entertainment	66%
Business	66%
Books & References	66%
LifeStyle	66%
Music & audio	33%
Personalization	66%
Puzzles	66%
Tools	33%
Travel & local	66%

with the inability of the app to deliver the necessary functionality (e.g., evoking unknown errors) – a view enforced by the TRR ranking scheme. Using an online sampling calculator (www.surveysystem.com/sscalc.htm), we randomly selected a sample of 42 reviews for further analyses. We found, in this sample, 30 reviews were supportive, while 12 reviews were critical. The numbers of positive reviews are much more than the critical reviews, thus, providing a high rating for the app under consideration. Hence, at least in the

case of this particular app, although there is a disparity between the user rankings and the TRR ranking, an experienced user would tend to agree with the TRR ranking.

Bad to Good: In this case, the users rated the app as the last and the TRR algorithm ranked it as the best. A few sample supportive (first two comments) and critical user comments (last two comments) for such a case from the category of Entertainment include:

- "I love this app! If you guys could just make it so that there's a messenger fake call setup, then it would be 5 stars!"
- "I loooove dis app!! I totally created fake messenger profiles of celebs and showed them to my friend who does not have Facebook or Messenger and Lol it actually worked she fell for it sooooo harddd!!! Thx for making this great app I love dis app!!!"
- "Please **add an insta account option** in which there should be a fake insta account i can show my bio and also other feature and prank a person that i hacked your account."
- "This app should not be here. One of my friends was bashed severely because of a prank. As creators of something you must **remember people can misuse this**. You may have made this for light pranks but you have to think the negative side as well."

As seen from above, the supportive reviews seem to focus on functional aspects (e.g., allowing to create fake profiles) – a view enforced by the TRR ranking scheme. On the other hand, the critical reviews seem to highlight the issues with the inability of the app to deliver add-on features (e.g., adding Instagram account) or ethical usages (e.g., misusing the app). For a randomly selected sample size of 42, the number of negative reviews are much than the supportive reviews (the number of good reviews is 15 and the number of bad reviews is 27), thus, providing a low rating for the app under consideration. Hence, in the case of this particular app, although there is a disparity between the user rankings and the TRR ranking, a user who is focused on the functional view of the app (i.e., is the app delivering what it promises to deliver) would tend to agree with the TRR ranking.

In a similar way, for 30 heterogeneous apps from top ten categories, we found seven apps that show opposite orderings. These seven cases again are grouped into two good to bad and bad to good categories:

Good to Bad: Two supportive and two critical user comments for such a case from the category of Travel & Local include:

- "Good app, while I was traveling in side train. the location of indication station almost correct but **PNR updates very slow**"
- "The best part of this application is that it works in offline mode. Now available in all 8 necessary languages, especially in hindi and bengali, very good application."
- "**Live status was not updated**... My train has moved and crossed over 5 big stations but its showing in the same boarded place itself."
- "**Now Not able search pnr stat**us, always showing Indian railways server error, while server is working fine in other app. Please correct it then it's become very useful as before."

As seen from above, again, the supportive reviews seem to focus on non-functional aspects (e.g., multi-lingual), while the critical reviews seem to highlight the issues with the inability of the app to deliver the necessary functionality (e.g. incorrect live status) – a view enforced by the TRR ranking scheme. For a randomly selected sample size of 42, the number of positive reviews are much than the supportive reviews (the number of good reviews is 28 and the number of bad reviews is 14), thus, providing a low rating for the app under consideration. Hence, in the case of this particular app, although there is a disparity between the user rankings and the TRR ranking, a user who is focused on the functional view of the app (i.e., is the app delivering what it promises to deliver) would tend to agree with the TRR ranking

Bad to Good: Two supportive and two critical comments for such a case from the category of Business include:

- "I am getting interviewed after applying and similar exp beside my frnd have regular oppertunities because he has paid service of naukri. I dont know how but some times i feel its because of paid or non paid."
- "One of the best app. Notifications are well organized. I wish if dummy openings can be removed which misguide the applicants and they don't get call despite of matching JD."
- "**App is good. But d employers in it r cheat**. I got a call from a Company named as 'Genius Solutions'. They called me for interview at their office in kirti nagar. They deposited 500 rs money from me and said by calling they will tell me the nearby location to my house. Now, its been one week they hvnt even called me n when i tried to call they r not responding. Beware guys n girls."
- "**This App is good no doubt** but one of the recruiter call me for the Interview & I had given the simple Interview & told I'm Selected & ask me to pay 750 as it is refundable. After paying it they again ask to pay 2000 for the training which is not refundable then I found that they are fraud & fooling me.. Then I decided not to go & pay 2000 for training. But My **750 is gone & very upset**."

As seen from above, the supportive reviews seem to focus on functional aspects (e.g., a good organization of notations) – a view enforced by the TRR ranking scheme. On the other hand, the critical reviews seem to highlight the issues with the external features (e.g., bad behavior by recruiters). For a randomly selected sample size of 42, the number of negative reviews are much than the supportive reviews (the number of good reviews is 11 and the number of bad reviews is 31), thus, providing a low rating for the app under consideration. Hence,

again, in the case of this particular app, although there is a disparity between the user rankings and the TRR ranking, a user who is focused on the functional view of the app (i.e., is the app delivering what it promises to deliver) would tend to agree with the TRR ranking.

V. Conclusion and Future Works

In this paper, we have described the technique to model and quantify the trust of software apps based on the programmatic (internal) view. The proposed technique, called TRR approach, also provides methods to analyze, and aggregate, internal views of software apps and use them to perform trust-based rating and ranking. We applied TRR approach to a few apps from popular categories and compared the rankings based on the user reviews with our rankings. Although, we found is most cases, the two rankings were different, a closer investigation does reveal that there are many similarities between the rankings, if the user is focused on the promised functional features of the app. As many users are not focused on the functional aspects only but give importance to other aspects (e.g., additional feature or look and feel), it is necessary to merge the internal view provided by the TRR approach with the external view obtained by more thorough investigation (e.g., sentiment analysis and reputation of users) of user reviews. Such a combined alternative will provide a holistic view of apps and their rankings – one of the future investigations we are planning to pursue. Other future efforts include applying the TRR approach to larger and diverse datasets and the exploration of other prevalent techniques (e.g., model checking) to compute the internal view.

References

[1] Number of apps available in leading app stores as of March 2017: www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[2] Why You Shouldn't Trust App Store Reviews (and What to Trust Instead): lifehacker.com/why-you-shouldnt-trust-app-store-reviews-and-what-to-1515379780.

[3] L. Gallege, D. U. Gamage, J. H. Hill, and R. R. Raje, "Understanding the trust of software-intensive distributed systems," Concurrency and Computation Practice and Experience, Volume 28, Issue 1, pp. 114-143, January 2016.

[4] L. Gallege, and R. R. Raje, "Parallel Methods for Evidence and Trust-basd Selection and Recommendation of Software Apps from Online Marketplaces," Proceedings of the 12th Annual Cyber and Information Security Research Conference, 2017.

[5] L. Gallege, "Trust-based Service Selection and Recommendation for Online Software Marketplaces (TruSStReMark)," PhD Thesis, IUPUI, December 2016.

[6] G. Shafer, "A Mathematical Theory of Evidence," Whitepaper, Princeton University Press, 1976.

[7] N. Indurkhya and F. J. Damerau, Handbook of Natural Language Processing, 2nd ed., Chapman and Hall Inc., 2010.

[8] Espresso:google.github.io/android-testing-supportlibrary/docs/espresso/index.html

[9] Monkey: developer.android.com/studio/test/monkey.html

[10] X. Ma, N. Wang, P. Xie, J. Zhou, X. Zhang, C. Fang, "An Automated Testing Platform for Mobile Applications," IEEE International Conference on Software Quality, Reliability and Security Companion, 2016.

[11] A. Machiry, R. Tahiliani, and M. Naik. "Dynodroid: An input generation system for android apps," In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 224–234, ACM, 2013.

[12] R. Mahmood, N. Mirzaei, and S. Malek, "Evodroid: Segmented evolutionary testing of android apps," In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 599–609, ACM, 2014.

[13] W. Choi, G. Necula, and K. Sen, "Guided gui testing of android apps with minimal restart and approximate learning," In ACM SIGPLAN Notices, volume 48, pp. 623–640. ACM, 2013.

[14] FindBugs: findbugs.sourceforge.net/

[15] Jlint: jlint.sourceforge.net/

[16] H. Khalid, M. Nagappan, A. E. Hassan, "Examining the Relationship between FindBugs Warnings and App Ratings," IEEE SOFTWARE, pp. 34–39, July/August 2016.

[17] S. Panichella, A. Di Sorboy, E. Guzmanz, C. A.Visaggioy, G. Canforay, and H. C. Gall, "How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution," ICSME, Bremen, Germany, pp. 281 -290, 2015.

[18] B. Pang, L. Lee, and S. Vaithyanathan, "ThumbsUp? Sentiment Classification using Machine Learning Techniques," Proceedings of the Empirical Methods in Natural Language Processing (EMNLP), 2002.

[19] F. Palomba, M. Linares-V´asquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "User Reviews Matter! Tracking Crowdsourced Reviews to Support Evolution of Successful Apps," ICSME, Bremen, Germany, pp. 291-300, 2015.

[20] M. Siegler, "YouTube Comes To A 5-Star Realization: Its Ratings Are Useless," Techcrunch, Sep 22, 2009.

[21] D. Hovemeyer and W. Pugh, "Finding Bugs Is Easy," ACM SIGPLAN Notices, vol. 39, no. 12, 2004.

[22] Evaluation-of-findbugs: girasoleyang.blogspot.com/2013/11/evaluation-of-findbugs-final-blogpost.html

[23] A. Jøsang, "Subjective Logic: A formalism for reasoning under uncertainty," Springer Verlag, 2016.

[24] A. Jøsang, "An Algebra for Assessing Trust in Certification Chains," NDSS Symposium, 1999.

[25] Top 10 Google Play categories: www.appbrain.com/stats/android-market-app-categories

[26] D. Ceolin, P. Groth, and W. Robert Van Hage, "Calculating the trust of event descriptions using provenance," Proceedings Of The SWPM, 2010.

[27] H.Wenchang Shi, Z. Liang, and B. Liang, "Using new fusion operations to improve trust expressiveness of subjective logic," Wuhan University Journal of Natural Sciences, Volume 16, number 5, Sep 2011.

[28] [28] M. G. Kendall, "A New Measure of Rank Correlation," Biometrika, Volume 30, Issue 1-2, pp. 81–93, 1938.